MäK Technologies
185 Alewife Brook Parkway
Cambridge, MA 02138

SBIR OSD97-003 Final Report

Integrating HLA into the Spearhead Game

Contract #M67004-97-C-0046

MäK Technologies Technical Report #TR-OSD97003-980515.1

Principal Investigator: Robert Faucher

Date of Submission: May 15, 1998

Submitted to: U.S Army Simulation, Training and Instrumentation Command

Table of Contents

Section 1 ..... **4**

1.1 Introduction 4

1.2 Summary of Results 5

1.3 Conclusion 6

Section 2 ..... **7**

2.1 Proposal Tasks 7
2.1.1 Integrate VR-Link 3.x into Spearhead 7
2.1.2 Re-Code all Spearhead-Specific DIS PDUs to HLA Interactions 8
2.1.2.1 Lobbying Interactions 8
2.1.2.2 IVIS Interactions 9
2.1.2.3 Re-Coding of Game Shortcuts to Reduce Network Bandwidth 10
2.1.3 Integrate VR-Link 3.x into existing Lobby Server 12
2.1.4 Design and build new Lobby Server functions which use RTI Services 12
2.1.5 Design Real-Time RTI for Game Market 12
2.1.6 Final Report 12

Section 3 Appendix A – Source Code API Samples **13**

3.1 Spearhead-Specific Interactions 13
3.1.1 SphChatInteraction 13
3.1.2 SphCommentInteraction 14
3.1.3 SphCtrlIVISAirAttackInteraction 16
3.1.4 SphCtrlIVISAirReconInteraction 17
3.1.5 SphCtrlIVISArtilleryInteraction 18
3.1.6 SphCtrlIVISEntityEngageInteraction 20
3.1.7 SphCtrlIVISEntityKillLabelInteraction 21
3.1.8 SphCtrlIVISEntityLabelInteraction 22
3.1.9 SphCtrlIVISEntityMoveToInteraction 24
3.1.10 SphCtrlIVISEntityOrientationInteraction 25
3.1.11 SphCtrlIVISEntityROEInteraction 26
3.1.12 DtMachineGunFireInteraction 28
3.1.13 DtMachineGunDetonationInteraction 29
3.1.14 GntByteMoverInteraction 32
3.1.15 GntControlMessageInteraction 33

3.2 Game Entity Identifier Conversion Functions 34

Section 4 Appendix B Federation File modifications **35**

Section 5 Installing the HLA Spearhead Demo CD **37**

5.1 System Requirements 37

5.2 Installation Steps 37

Section 6 Section 6 Running the HLA Spearhead Demo **40**

6.1 Running the RTI Exec 40

6.2 Running HLA Spearhead 40

6.2.1 Run A Single Player Game First To Test HLA Spearhead 41

6.2.2 Playing a Simple Multiplayer Game 41

6.2.2.1 Instructions for the Host Machine 42

6.2.2.2 Instructions for the Joiner Machine 42

6.3 Known Problems Running HLA Spearhead Phase 1 43

MaK Technologies
Integrating HLA into the Spearhead Game
Contract No. M67004-97-C-0046
May 11, 1998

Section 1

1.1 Introduction

As personal computers grow in power and speed and hardware costs decline, complex 3D simulations that once required multi-million dollar systems are now capable of running on mass market personal computer systems. The exponential growth of the internet in the last few years has led to a substantial number of commercial games and simulations that use the internet to link together remote machines. Moreover, the proliferation of Local Area Networks (LANs) in business and academic institutions has led to an increasing demand for commercial games and simulations that can run on a LAN topology. Presently there exists no standard for a simulation or game networking protocol and architecture in the commercial sector. Typically most game and simulation manufacturers will create their own set of protocols and networking architectures for their products. As a result, there are countless solutions in place for linking together various commercial games and simulations.

HLA shows promise for creating a unified standard networking architecture that can be deployed in the commercial sector. Currently HLA has only gained widespread acceptance within the Department of Defense. Just as the interoperability of various simulations is promised by HLA within DOD, the same level of interoperability can be brought into the commercial sector.

The core challenge of this project was to make a non-military application that uses HLA and helps promote HLA in non-military markets. If HLA is to survive and flourish, the more industries that have heard of HLA, use HLA successfully, and acknowledge what it offers, the more commercial hardware and software vendors will develop tools which implement and adhere to the HLA standard. This project defines a path to accomplish this goal in the video game market with a high likelihood of success.

The original networking architecture of commercial Spearhead was based on DIS-Lite, which is a version of DIS with reduced network bandwidth intended for PC games. Although DIS-Lite requires less bandwidth than DIS, its architecture remains fundamentally the same.

The primary goal of this project was to integrate HLA with Spearhead, MAK's completed, state of the art, M1A2 Abrams tank game. The game was originally written to use the DIS-Lite and VR-Link 2.4.x networking protocol and architecture developed by MAK technologies. At the commencement of this effort, Spearhead was in Beta form and all networking functionality was virtually complete. The non-HLA version of Spearhead permitted players to engage in distributed interactive games over LANs, the Internet, NULL modems and Dial up modems.

A secondary goal of this project was to convert MAK's existing Spearhead-specific lobby server into an HLA application. A "lobby server" is an application that runs on a dedicated machine connected to the internet via a T1 line. The Spearhead lobby server is a C++ application written at MAK. The existing lobby server was not based on any standardized protocol or architecture, such as DIS, instead MAK created its own custom lobby server protocols. This means the lobby server would need to be re-written to use HLA and the RTI.

The primary purpose of a lobby server is to provide a point of entry for players to join together in a simulation. For example, if a player wants to "host" a game s/he connects to the lobby server using Spearhead and creates a hosted game that other players may join. A prospective player would then connect to the lobby server and see a list of games being hosted. The player can then join a hosted game and choose a team (i.e., red or blue) to join. While games are being hosted, players can chat with one another, enter and leave hosted games and disconnect. Once the host is satisfied that s/he has enough players for their game, the host starts the simulation. At that point the game's mission file is uploaded from the lobby server to all the players in the game. Once all players have successfully received the mission file, the players are now communicating peer to peer and are disconnected from the lobby server.

The final goal was to create a new real-time RTI specifically aimed at the game market. This RTI would allow users to communicate with the new RTI over the internet using dedicated lines or dial up modems through a commercial Internet Service Provider (ISP).

1.2 Summary of Results

The primary goal of integrating HLA with Spearhead was achieved. VR-Link 2.4.x and DIS-Lite were replaced by VR-Link 3.x, the DMSO RTI (v1.0 release 3) and a modified VR-Link 3.x federation (FED) file. Nearly all the functionality of the DIS-Lite version was ported over to HLA. Presently, HLA Spearhead will run existing spearhead-specific missions in single or multiplayer mode on a LAN. Essentially, we have a commercial grade 3D tank simulation that uses HLA for single and multiple federates. Then end result of this effort demonstrates that it is possible to create a commercial grade computer game using HLA.

The secondary goal of converting MAK's internet lobby server to HLA was not realized. The principle reason was the amount of time and resources required to convert Spearhead to HLA. The original estimates proved inaccurate owing to a number of unforeseen problems and side effects resulting from the conversion effort and the integration of VR-Link 3.x. Moreover, the DMSO RTI is incapable of running on the internet, thus MAK would have had to create an internet capable RTI in advance.

However, the lobbying functionality required for playing LAN games was converted to use custom HLA interactions. Many of these lobbying interactions could be used to create an internet lobby server. It is worth noting that the fundamental functions and architecture of LAN lobbying are identical to internet lobbying (i.e., listing hosted missions, enumerating players, chatting, etc). In LAN mode, one of the federates becomes a "temporary" lobby server until all the other federates have joined and the exercise is in progress.

Finally, a new real-time RTI specifically created for the game market was not created. There are two reasons for this:

- The time required for the conversion of Spearhead to HLA and the LAN lobbying functionality required to play multiplayer games took longer than expected.
- During course of this effort, MAK has developed its own real-time RTI that will be eventually modified to support internet connectivity.

1.3 Conclusion

The end results of Phase I demonstrate that HLA is a viable architecture for developing commercial grade games and simulations. HLA has the functionality, flexibility, speed and power to allow a wide variety of commercial grade simulations to be developed and deployed for the mass market.

Having the benefit of hindsight, it can be argued that it is easier and faster to develop commercial games and simulations using VR-Link 3.x with modified FOM and FED files instead of using VR-Link 2.4.x with game specific changes to DIS. The flexibility and ease of use of the FED and FOM files makes it relatively easy to customize and develop simulations based on the RPR FOM. Moreover, since VR-Link 3.x hides many of the RTI specific details, adding and modifying new HLA interactions is easy.

One of the downsides of this effort were the problems in converting a DIS-Lite application to HLA. Although VR-Link 3.x hides many of the implementation details of DIS versus HLA from the user, there were still many lines of code that required modifications on account of Spearhead-specific deviations from the standard DIS architecture.

Before HLA can be deployed and accepted in the commercial sector and new game-specific RTI must be developed. The existing DMSO RTI only runs on LANs and the RTIEXEC is command line application. This creates some problems that mitigate against the use of the current RTI in the commercial game/simulation market:

- Any commercial multiplayer game **must** run on the internet. Failure to do so is tantamount to market suicide.

- Any commercial multiplayer game running on the internet must support direct connections, such as T1 lines as well as dial up modems via an ISP.

- A commercial game or simulation needs to appear seamless to the end user. This means no game publisher is ever likely to accept an RTIEXEC running as a stand alone application inside a DOS window within Windows 95 or NT. In other words, the RTIEXEC should be invisible and hidden from the user.

However, since MAK now has its own real-time RTI, we can modify it to suit the above requirements. Effectively, there are no technical barriers that prevent HLA from being a viable and powerful architecture to use for the development of commercial online games and simulations.

Section 2

2.1 Proposal Tasks

This section will cover our work on each of the tasks described in the Phase I proposal. Each section includes a brief discussion of our work on the task along with references to the documentation and additional materials resulting from the task.

2.1.1 Integrate VR-Link 3.x into Spearhead

Since the existing Spearhead application was based on VR-Link 2.4.x and DIS-Lite, we had to remove DIS-Lite and use VR-Link 3.x instead. This is where the bulk of the work in this project was expended. Since the API from VR-Link 2.4.x to VR-Link 3.x had a number of significant changes in structures and classes and the API, the Spearhead code required corollary modifications. Since many of these classes and structures were used throughout the code, there were literally thousands of lines of code that required minor modifications. Examples of these structures and classes include very common math structures such vectors, matrixes, etc.

The following are examples of such changes. Note the use of the SPH_DIS_BUILD compiler directive to differentiate the DIS version from the HLA version.

```
#ifndef SPH_DIS_BUILD
  atVec.z = spReal(-cos(simAngles.psi));
  atVec.y = spReal(0.);
  atVec.x = spReal(-sin(simAngles.psi));
#else
  atVec.z = spReal(-cos(simAngles.psi()));
  atVec.y = spReal(0.);
  atVec.x = spReal(-sin(simAngles.psi()));
#endif
```

```
#ifndef SPH_DIS_BUILD
  DtSetVec(m_gravityVector, 0.0, 0.0, -KtGRAVITY_CONSTANT);
#else
  m_gravityVector = DtVector(0.0, 0.0, -KtGRAVITY_CONSTANT);
#endif
```

```
#ifndef SPH_DIS_BUILD
void GtSimBody::MeToWorld(DtDcm meToWorld, int quickFlag)
#else
void GtSimBody::MeToWorld(DtDcmRef meToWorld, int quickFlag)
#endif
```

```
#ifndef SPH_DIS_BUILD
        return m_draEntity->DtAcceleration;
#else
        return m_draEntity->entityStateRep()->acceleration();
#endif // SPH_DIS_BUILD
```

For the commercial version of Spearhead, a number of callback functions were coded for the management of events such gun fire, detonate, burst, collisions, etc. Each of those callback functions required some level of modification in order to implement their HLA equivalents. This is a normal side effect of converting DIS PDUs to HLA Interactions.

2.1.2 Re-Code all Spearhead-Specific DIS PDUs to HLA Interactions

Since Spearhead was designed to be a game for the commercial sector, it was necessary to code customized DIS PDUs for some of the game's features. The two main areas requiring custom PDUs were the game lobbying, including chat features, and the IVIS.

2.1.2.1 Lobbying Interactions

As mentioned, the lobbying component of Spearhead permits players on LANs and the internet to join together to play a hosted game. Some of the LAN lobbying functions were implemented as custom DIS PDUs, naturally they required to be converted to HLA Interactions before a multiplayer game could even be tested in HLA Spearhead. These lobbying functions include:

- **Chat**. During the lobbying phase it is imperative that players have the ability to chat with one another. The chat feature allows a player to enter a text message which is then sent to other players on the system. Chat can be filtered to control message flow. For example, Spearhead implements "private" and "public" chat filters that allow the player to select whether the message is sent to only one selected player or to all players.

- **Mission File Transfer.** Since Spearhead uses mission script files for every simulation that is run, all players must have the same mission file before the simulation can commence. In a LAN lobbied game, the mission is sent from the game "host" to all the players that have joined. In other words, the "host" becomes a de-facto temporary server and the joiners are clients. The mission file is then sent to all the players once the host elects to start the game. The Internet case is somewhat different since the lobby server has a copy of the host's mission file in memory. Once the host elects to start the game, the lobby server sends the mission file to all the players. For Phase I, we implemented the temporary file server and file transfer functions for LAN games.

- **Mission Briefing Transfer.** This is similar to the file transfer functions noted above. In this case, the joiner can request a mission briefing from the host's machine. The mission briefing contains statistics about the mission (i.e., number and kind of friendly/enemy entities, number of player controlled vehicles on each team) as well as a few paragraphs of text explaining the mission goals. The importance of this feature

Requests that an entity is to be user-labeled (string) on the IVIS. The player selects an entity to label and can assign it either an iconic representation (i.e., Main Battle Tank) or a text label (e.g.,"Red Charlie"). This update will be sent to all the players on the team. The remote players will receive an interaction to update both the icon and text label.

SphCtrlIVISEntityKillLabelInteraction

Request that a specific IVIS entity label (string) should be removed. The player selects an entity to un-label. The selected entity's icon and text label will be deleted and the entity will be represented by a question mark "?" icon to indicated an unknown entity. This update will be sent to all the players on the team and their representation of the selected entity will be marked as unknown.

SphCtrlIVISEntityMoveToInteraction

Requests a specific entity to move to a given coordinate. The player selects a friendly entity and a location. If the friendly entity is being controlled being artificial intelligence, then the entity will move to the specified location. If the friendly entity is a player controlled vehicle, then a chat message of the form: *"Move To at 123.89, 84.22"* will be displayed to the player.

SphCtrlIVISEntityOrientationInteraction

Requests a specific entity to orient its chassis to a given angle. The player selects a friendly entity and then a point on the map to orient towards. If the friendly entity is being controlled being artificial intelligence, then the entity will orient itself towards the specified point. If the friendly entity is a player controlled vehicle, then a chat message of the form: *"Set Orientation To 270"* will be displayed to the player.

SphCtrlIVISEntityROEInteraction

Requests a specific entity to change its rules of engagement. The player selects a rule of engagement such as, Weapons Tight, and then selects a friendly entity. If the friendly entity is being controlled being artificial intelligence, then the entity will change its rules of engagement. If the friendly entity is a player controlled vehicle, then a chat message of the form: *"Set Rule of Engagement to Weapons Tight"* will be displayed to the player.

2.1.2.3 Re-Coding of Game Shortcuts to Reduce Network Bandwidth

In the commercial version of Spearhead, we made several shortcuts to reduce network bandwidth. For example, we shortened the DIS entity identifier to use only a player number and an entity number, both represented as unsigned 16 bit integers. Since the RTI uses its own entity ID scheme, with HLA entities represented as unique unsigned 32 bit integers, we needed to create functions and macros to facilitate the conversion to HLA. In order to preserve as much as the code as possible, we created functions to translate an HLA ID to a Spearhead Game Entity Identifier and vice-versa.

One problem is that commercial Spearhead would create the Game Entity Identifiers based on the number of players and entities in the exercise. Normally the RTI is supposed to generate these ID's for the federates, however this scheme would have severe side effects on the remote entity identifier paradigms used in Spearhead. For example, a DIS-Lite DtGameEntityIdentifier is constructed based on the following rules:

The Player Identifier is based on the nth player in the game with the host always being player ID 1 and successive players joined in the exercise are 2,3, and so on. In single player games, the Player ID is always 1.

The entity number is based on the nth entity being simulated by a local machine (federate). The first entity being locally simulated (the player controlled vehicle) will be 1, the next will be 2 and so on. Since Spearhead is based on a distributed simulation scheme, simulation of AI controlled entities is divided equally amongst all players in a federation (mission).

Consider a simple case of a two player game simulating 2 player controlled M1A2s and two AI controlled T-72s. The following DtGameEntityIdentifiers would be generated:

Entity	Player Identifier	Entity Number
Player 1 M1A2	1	1
Player 2 M1A2	2	1
T-72 #1	1	2
T-72 #2	2	2

Since the DtGameEntityIdentifiers are algorithmically derived, we needed to mimic the same algorithm to generate HLA identifiers. Fortunately version 1.0 of the RTI allows the application to specify its own HLA identifiers rather than relying on the RTI to generate them. Within VR-Link 3.x, we know that the RTI also bases unique identifiers on the number of players and entities. Using the above rules we can generate the following HLA Identifiers for our simple two player game:

Entity	HLA Identifier
Player 1 M1A2	100001
Player 2 M1A2	200001
T-72 #1	100002
T-72 #2	200002

The solution to maintaining the code was to have functions that transparently convert from the DIS-Lite identifier scheme to the HLA identifier scheme. These functions were significant since entity hashing keys are all based on the game generated entity identifiers. In the commercial version of Spearhead, we use the tuple DtGameEntityIdentifier to perform lookups in the HLA version we use an unsigned 32 bit integer as the hash key. Naturally, there were many places in the code that required that the identifiers were massaged to use the conversion functions.

Note: DtGameEntityIdentifier was defined a tuple for DIS-Lite implementation and as a 32 bit unsigned integer for HLA implementation.

The following code fragments illustrate how these conversions were accomplished.

```
GtSimEntity* GtSimEntityList::LookupEntity(DtGameEntityIdentifier* id)
{

 if (!id)
       return NULL;

#ifndef SPH_DIS_BUILD
 return (GtSimEntity *)lookup(id);
#else
 DtHashKey key((int)Cv2OID(*id));
 return (GtSimEntity *)lookup(key);
#endif
}
```

```
m_collisionPdu->setIssueId( Cv2OID( *GetId() ) );
m_collisionPdu->setCollideId( Cv2OID( *entity->GetId() ) );
```

```
DtGameEntityIdentifier attackerId = Cv2GEId( detonatePdu->attackerId() );
DtGameEntityIdentifier targetId = Cv2GEId( detonatePdu->targetId() );
```

2.1.3 Integrate VR-Link 3.x into existing Lobby Server

No work done on this task since there was no time left over from the conversion of Spearhead from DIS-Lite to HLA..

2.1.4 Design and build new Lobby Server functions which use RTI Services

No work done on this task since there was no time left over from the conversion of Spearhead from DIS-Lite to HLA..

2.1.5 Design Real-Time RTI for Game Market

MAK Technologies now has its own RTI that will be customized for the game market. The development of this RTI took place independently of this SBIR.

2.1.6 Final Report

This report along with appendices and attachments detail our work for this Phase I SBIR.

Section 3 Appendix A – Source Code API Samples

3.1 Spearhead-Specific Interactions

3.1.1 SphChatInteraction

```
typedef void (*SphChatInteractionCb)(SphChatInteraction* inter, void* usr);

class SphChatInteraction : public DtInteraction
{
public:
 SphChatInteraction();

 SphChatInteraction(const RTI::ParameterHandleValuePairSet& pvList,
       RTI::RTIambassador* amb);

 SphChatInteraction(const DtGameEntityIdentifier &sender,
       const DtGameEntityIdentifier &receiver,
       LPCTSTR lptszMessage,
       GtTeamType filter = GtTeamNeutral);

 // copy constructor
  SphChatInteraction(const SphChatInteraction& orig);

 virtual ~SphChatInteraction();

 // assignment operator
  SphChatInteraction& operator=(const SphChatInteraction& orig);

 //get the paramater handle value pair set for the interaction
 const RTI::ParameterHandleValuePairSet& phvps(
       RTI::RTIambassador* amb) const;

 //set the values from parameter value list
 virtual void setFromPhvps(const RTI::ParameterHandleValuePairSet& pvList,
       RTI::RTIambassador* amb);

 //get the interaction class handle
 virtual RTI::InteractionClassHandle interactionClassHandle(
       RTI::RTIambassador* amb) const;

 //get the number of parameters
 virtual int numParameters() const;

 //name of the interaction
 virtual const char* name() const;

 //print the interaction's data
 virtual void printData() const;

 //accessors
 LPCTSTR GetChatMessage() const;
  GtTeamType TeamFilter() const;
 virtual const DtGameEntityIdentifier& SenderId() const;
 virtual const DtGameEntityIdentifier& ReceiverId() const;
 virtual const DtObjectId& HLASenderId() const;
 virtual const DtObjectId& HLAReceiverId() const;
```

```
 //mutators

 int  SetChatMessage( LPCTSTR lptszMessage, GtTeamType filter = GtTeamNeutral );
 void SetTeamFilter(GtTeamType team);
 virtual void SetSenderId(const DtGameEntityIdentifier &id);
 virtual void SetReceiverId(const DtGameEntityIdentifier &id);
 virtual void SetHLASenderId(const DtObjectId &id);
 virtual void SetHLAReceiverId(const DtObjectId &id);

public:
 //get the interacton class handle
 static RTI::InteractionClassHandle classHandle(
       RTI::RTIambassador* amb);

 //create a SphChatInteraction.  Caller responsbile for deletion
 static DtInteraction* create(RTI::InteractionClassHandle h,
       const RTI::ParameterHandleValuePairSet& p,
       RTI::RTIambassador* amb);

 // Add (register)/Remove (deregister) function to be called
 // when the interaction occurs.
 static void addCallback(DtExerciseConn* conn,
       SphChatInteractionCb cb, void* usr);

 static void removeCallback(DtExerciseConn* conn,
       SphChatInteractionCb cb, void* usr);

 LPCTSTR sphStrnCpy( LPTSTR lptszDest, LPCTSTR lptszSrc, size_t count );

protected:
 //our actual data to be sent/received
 GtTeamType    m_TeamFilter;
 TCHAR      m_tszChatMessage[MAX_CHAT_MSG_LEN + 1];
 DtGameEntityIdentifier     m_SenderId;
 DtGameEntityIdentifier     m_ReceiverId;
 DtObjectId                 m_HLASenderId;
 DtObjectId                 m_HLAReceiverId;


protected:
 //used as return value by phvps - does not store state
 static RTI::ParameterHandleValuePairSet* thePvList;
};
```

3.1.2 SphCommentInteraction

```
typedef void (*SphCommentInteractionCb)(SphCommentInteraction* inter, void* usr);

class SphCommentInteraction : public DtInteraction
{
public:
 SphCommentInteraction();

 SphCommentInteraction(const RTI::ParameterHandleValuePairSet& pvList,
       RTI::RTIambassador* amb);
```

```
SphCommentInteraction(const DtGameEntityIdentifier &sender,
        const DtGameEntityIdentifier &receiver,
        LPCTSTR lptszMessage,

        GtTeamType filter = GtTeamNeutral);


virtual ~SphCommentInteraction();

//get the paramater handle value pair set for the interaction
const RTI::ParameterHandleValuePairSet& phvps(
        RTI::RTIambassador* amb) const;

//set the values from parameter value list
virtual void setFromPhvps(const RTI::ParameterHandleValuePairSet& pvList,
        RTI::RTIambassador* amb);

//get the interaction class handle
virtual RTI::InteractionClassHandle interactionClassHandle(
        RTI::RTIambassador* amb) const;

//get the number of parameters
virtual int numParameters() const;

//name of the interaction
virtual const char* name() const;

//print the interaction's data
virtual void printData() const;

//accessors
LPCTSTR GetChatMessage() const;
  GtTeamType TeamFilter() const;
virtual const DtGameEntityIdentifier& SenderId() const;
virtual const DtGameEntityIdentifier& ReceiverId() const;

//mutators
int  SetChatMessage( LPCTSTR lptszMessage, GtTeamType filter = GtTeamNeutral );
void SetTeamFilter(GtTeamType team);
virtual void SetSenderId(const DtGameEntityIdentifier &id);
virtual void SetReceiverId(const DtGameEntityIdentifier &id);

public:
//get the interacton class handle
static RTI::InteractionClassHandle classHandle(
        RTI::RTIambassador* amb);

//create a SphCommentInteraction.  Caller responsbile for deletion
static DtInteraction* create(RTI::InteractionClassHandle h,
        const RTI::ParameterHandleValuePairSet& p,
        RTI::RTIambassador* amb);

// Add (register)/Remove (deregister) function to be called
// when the interaction occurs.
static void addCallback(DtExerciseConn* conn,
        SphCommentInteractionCb cb, void* usr);

static void removeCallback(DtExerciseConn* conn,
        SphCommentInteractionCb cb, void* usr);

LPCTSTR sphStrnCpy( LPTSTR lptszDest, LPCTSTR lptszSrc, size_t count );
```

```
protected:
 //our actual data to be sent/received
 GtTeamType    m_TeamFilter;
 TCHAR     m_tszChatMessage[MAX_CHAT_MSG_LEN + 1];
 DtGameEntityIdentifier      m_SenderId;

 DtGameEntityIdentifier      m_ReceiverId;


protected:
 //used as return value by phvps - does not store state
 static RTI::ParameterHandleValuePairSet* thePvList;
};
```

3.1.3 SphCtrlIVISAirAttackInteraction

```
typedef void (*SphCtrlIVISAirAttackInteractionCb)(SphCtrlIVISAirAttackInteraction* inter, void* usr);

class SphCtrlIVISAirAttackInteraction : public DtInteraction
{
public:
 SphCtrlIVISAirAttackInteraction();

 SphCtrlIVISAirAttackInteraction(const RTI::ParameterHandleValuePairSet& pvList,
       RTI::RTIambassador* amb);

 SphCtrlIVISAirAttackInteraction(GtTeamType  senderTeam,
       double x,
       double y);

 virtual ~SphCtrlIVISAirAttackInteraction();

 //get the paramater handle value pair set for the interaction
 const RTI::ParameterHandleValuePairSet& phvps(
       RTI::RTIambassador* amb) const;

 //set the values from parameter value list
 virtual void setFromPhvps(const RTI::ParameterHandleValuePairSet& pvList,
       RTI::RTIambassador* amb);

 //get the interaction class handle
 virtual RTI::InteractionClassHandle interactionClassHandle(
       RTI::RTIambassador* amb) const;

 //get the number of parameters
 virtual int numParameters() const;

 //name of the interaction
 virtual const char* name() const;

 //print the interaction's data
 virtual void printData() const;

 //accessors
 const GtTeamType senderTeam() const;
  const double X() const;
```

```
 //mutators
 void SetSenderTeam( const GtTeamType senderTeam );
   void SetX( const double x );
   void SetY( const double y );

public:
 //get the interacton class handle

 static RTI::InteractionClassHandle classHandle(

       RTI::RTIambassador* amb);

 //create a SphCtrlIVISAirAttackInteraction.  Caller responsbile for deletion
 static DtInteraction* create(RTI::InteractionClassHandle h,
       const RTI::ParameterHandleValuePairSet& p,
       RTI::RTIambassador* amb);

 // Add (register)/Remove (deregister) function to be called
 // when the interaction occurs.
 static void addCallback(DtExerciseConn* conn,
       SphCtrlIVISAirAttackInteractionCb cb, void* usr);

 static void removeCallback(DtExerciseConn* conn,
       SphCtrlIVISAirAttackInteractionCb cb, void* usr);

protected:
 //our actual data to be sent/received
 GtTeamType    m_SenderTeam;
 double        m_x;
 double        m_y;

protected:
 //used as return value by phvps - does not store state
 static RTI::ParameterHandleValuePairSet* thePvList;
};
```

3.1.4 SphCtrlIVISAirReconInteraction

```
typedef void (*SphCtrlIVISAirReconInteractionCb)(SphCtrlIVISAirReconInteraction* inter, void* usr);

class SphCtrlIVISAirReconInteraction : public DtInteraction
{
public:
 SphCtrlIVISAirReconInteraction();

 SphCtrlIVISAirReconInteraction(const RTI::ParameterHandleValuePairSet& pvList,
       RTI::RTIambassador* amb);

 SphCtrlIVISAirReconInteraction(GtTeamType senderTeam,
       double x,
       double y);

 virtual ~SphCtrlIVISAirReconInteraction();

 //get the paramater handle value pair set for the interaction
```

```
        RTI::RTIambassador* amb) const;

    //set the values from parameter value list
    virtual void setFromPhvps(const RTI::ParameterHandleValuePairSet& pvList,
        RTI::RTIambassador* amb);

    //get the interaction class handle
    virtual RTI::InteractionClassHandle interactionClassHandle(
        RTI::RTIambassador* amb) const;

    //get the number of parameters

    virtual int numParameters() const;


    //name of the interaction
    virtual const char* name() const;

    //print the interaction's data
    virtual void printData() const;

    //accessors
    const GtTeamType senderTeam() const;
     const double X() const;
     const double Y() const;

    //mutators
    void SetSenderTeam( const GtTeamType senderTeam );
     void SetX( const double x );
     void SetY( const double y );

  public:
    //get the interacton class handle
    static RTI::InteractionClassHandle classHandle(
        RTI::RTIambassador* amb);

    //create a SphCtrlIVISAirReconInteraction.  Caller responsbile for deletion
    static DtInteraction* create(RTI::InteractionClassHandle h,
        const RTI::ParameterHandleValuePairSet& p,
        RTI::RTIambassador* amb);

    // Add (register)/Remove (deregister) function to be called
    // when the interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        SphCtrlIVISAirReconInteractionCb cb, void* usr);

    static void removeCallback(DtExerciseConn* conn,
        SphCtrlIVISAirReconInteractionCb cb, void* usr);

  protected:
    //our actual data to be sent/received
    GtTeamType    m_SenderTeam;
    double        m_x;
    double        m_y;

  protected:
    //used as return value by phvps - does not store state
    static RTI::ParameterHandleValuePairSet* thePvList;
};
```

3.1.5 SphCtrlIVISArtilleryInteraction

```
typedef void (*SphCtrlIVISArtilleryInteractionCb)(SphCtrlIVISArtilleryInteraction* inter, void* usr);

class SphCtrlIVISArtilleryInteraction : public DtInteraction
{
public:
 SphCtrlIVISArtilleryInteraction();

 SphCtrlIVISArtilleryInteraction(const RTI::ParameterHandleValuePairSet& pvList,
       RTI::RTIambassador* amb);



 SphCtrlIVISArtilleryInteraction(GtTeamType  senderTeam,
       double x,
       double y);

 virtual ~SphCtrlIVISArtilleryInteraction();

 //get the paramater handle value pair set for the interaction
 const RTI::ParameterHandleValuePairSet& phvps(
       RTI::RTIambassador* amb) const;

 //set the values from parameter value list
 virtual void setFromPhvps(const RTI::ParameterHandleValuePairSet& pvList,
       RTI::RTIambassador* amb);

 //get the interaction class handle
 virtual RTI::InteractionClassHandle interactionClassHandle(
       RTI::RTIambassador* amb) const;

 //get the number of parameters
 virtual int numParameters() const;

 //name of the interaction
 virtual const char* name() const;

 //print the interaction's data
 virtual void printData() const;

 //accessors
 const GtTeamType senderTeam() const;
  const double X() const;
  const double Y() const;

 //mutators
 void SetSenderTeam( const GtTeamType senderTeam );
  void SetX( const double x );
  void SetY( const double y );

public:
 //get the interacton class handle
 static RTI::InteractionClassHandle classHandle(
       RTI::RTIambassador* amb);

 //create a SphCtrlIVISArtilleryInteraction.  Caller responsbile for deletion
 static DtInteraction* create(RTI::InteractionClassHandle h,
```

```
        RTI::RTIambassador* amb);

    // Add (register)/Remove (deregister) function to be called
    // when the interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        SphCtrlIVISArtilleryInteractionCb cb, void* usr);

    static void removeCallback(DtExerciseConn* conn,
        SphCtrlIVISArtilleryInteractionCb cb, void* usr);

protected:
    //our actual data to be sent/received
    GtTeamType    m_SenderTeam;
    double        m_x;
    double        m_y;


protected:

    //used as return value by phvps - does not store state
    static RTI::ParameterHandleValuePairSet* thePvList;
};
```

3.1.6 SphCtrlIVISEntityEngageInteraction

```
typedef void (*SphCtrlIVISEntityEngageInteractionCb)(SphCtrlIVISEntityEngageInteraction* inter, void* usr);

class SphCtrlIVISEntityEngageInteraction : public DtInteraction
{
public:
    SphCtrlIVISEntityEngageInteraction();

    SphCtrlIVISEntityEngageInteraction(const RTI::ParameterHandleValuePairSet& pvList,
        RTI::RTIambassador* amb);

    SphCtrlIVISEntityEngageInteraction(GtTeamType senderTeam,
        DtGameEntityIdentifier& entityID,
        DtGameEntityIdentifier& targetID);

    virtual ~SphCtrlIVISEntityEngageInteraction();

    //get the paramater handle value pair set for the interaction
    const RTI::ParameterHandleValuePairSet& phvps(
        RTI::RTIambassador* amb) const;

    //set the values from parameter value list
    virtual void setFromPhvps(const RTI::ParameterHandleValuePairSet& pvList,
        RTI::RTIambassador* amb);

    //get the interaction class handle
    virtual RTI::InteractionClassHandle interactionClassHandle(
        RTI::RTIambassador* amb) const;

    //get the number of parameters
    virtual int numParameters() const;
```

```
 virtual const char* name() const;

 //print the interaction's data
 virtual void printData() const;

 //accessors
 const GtTeamType senderTeam() const;
 const DtGameEntityIdentifier& entityID() const;
  const DtGameEntityIdentifier& targetID() const;

 //mutators
 void SetSenderTeam( const GtTeamType senderTeam );
 void SetEntityID( const DtGameEntityIdentifier& id );
  void SetTargetID( const DtGameEntityIdentifier& id );

public:
 //get the interacton class handle
 static RTI::InteractionClassHandle classHandle(
       RTI::RTIambassador* amb);



 //create a SphCtrlIVISEntityEngageInteraction.  Caller responsbile for deletion
 static DtInteraction* create(RTI::InteractionClassHandle h,
       const RTI::ParameterHandleValuePairSet& p,
       RTI::RTIambassador* amb);

 // Add (register)/Remove (deregister) function to be called
 // when the interaction occurs.
 static void addCallback(DtExerciseConn* conn,
       SphCtrlIVISEntityEngageInteractionCb cb, void* usr);

 static void removeCallback(DtExerciseConn* conn,
       SphCtrlIVISEntityEngageInteractionCb cb, void* usr);

protected:
 //our actual data to be sent/received
 GtTeamType                    m_SenderTeam;
 DtGameEntityIdentifier      m_entityID;
 DtGameEntityIdentifier      m_targetID;

protected:
 //used as return value by phvps - does not store state
 static RTI::ParameterHandleValuePairSet* thePvList;
};
```

3.1.7 SphCtrlIVISEntityKillLabelInteraction

```
typedef void (*SphCtrlIVISEntityKillLabelInteractionCb)(SphCtrlIVISEntityKillLabelInteraction* inter, void*
usr);

class SphCtrlIVISEntityKillLabelInteraction : public DtInteraction
{
public:
 SphCtrlIVISEntityKillLabelInteraction();
```

```
        RTI::RTIambassador* amb);

    SphCtrlIVISEntityKillLabelInteraction(GtTeamType senderTeam,
        DtGameEntityIdentifier& entityID);

    virtual ~SphCtrlIVISEntityKillLabelInteraction();

    //get the paramater handle value pair set for the interaction
    const RTI::ParameterHandleValuePairSet& phvps(
        RTI::RTIambassador* amb) const;

    //set the values from parameter value list
    virtual void setFromPhvps(const RTI::ParameterHandleValuePairSet& pvList,
        RTI::RTIambassador* amb);

    //get the interaction class handle
    virtual RTI::InteractionClassHandle interactionClassHandle(
        RTI::RTIambassador* amb) const;

    //get the number of parameters
    virtual int numParameters() const;


    //name of the interaction

    virtual const char* name() const;

    //print the interaction's data
    virtual void printData() const;

    //accessors
    const GtTeamType senderTeam() const;
    const DtGameEntityIdentifier& entityID() const;

    //mutators
    void SetSenderTeam( const GtTeamType senderTeam );
    void SetEntityID( const DtGameEntityIdentifier& id );

public:
    //get the interacton class handle
    static RTI::InteractionClassHandle classHandle(
        RTI::RTIambassador* amb);

    //create a SphCtrlIVISEntityKillLabelInteraction.  Caller responsbile for deletion
    static DtInteraction* create(RTI::InteractionClassHandle h,
        const RTI::ParameterHandleValuePairSet& p,
        RTI::RTIambassador* amb);

    // Add (register)/Remove (deregister) function to be called
    // when the interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        SphCtrlIVISEntityKillLabelInteractionCb cb, void* usr);

    static void removeCallback(DtExerciseConn* conn,
        SphCtrlIVISEntityKillLabelInteractionCb cb, void* usr);

protected:
    //our actual data to be sent/received
    GtTeamType              m_SenderTeam;
    DtGameEntityIdentifier  m_entityID;
```

```
 //used as return value by phvps - does not store state
 static RTI::ParameterHandleValuePairSet* thePvList;
};
```

3.1.8 SphCtrlIVISEntityLabelInteraction

```
typedef void (*SphCtrlIVISEntityLabelInteractionCb)(SphCtrlIVISEntityLabelInteraction* inter, void* usr);

class SphCtrlIVISEntityLabelInteraction : public DtInteraction
{
public:
 SphCtrlIVISEntityLabelInteraction();

 SphCtrlIVISEntityLabelInteraction(const RTI::ParameterHandleValuePairSet& pvList,
       RTI::RTIambassador* amb);

 SphCtrlIVISEntityLabelInteraction(GtTeamType senderTeam,
       DtGameEntityIdentifier&    entityID,
       GtBodyType entityBodyType,
       GtTeamType entityTeam,

       char* szLabel);


 virtual ~SphCtrlIVISEntityLabelInteraction();

 //get the paramater handle value pair set for the interaction
 const RTI::ParameterHandleValuePairSet& phvps(
       RTI::RTIambassador* amb) const;

 //set the values from parameter value list
 virtual void setFromPhvps(const RTI::ParameterHandleValuePairSet& pvList,
       RTI::RTIambassador* amb);

 //get the interaction class handle
 virtual RTI::InteractionClassHandle interactionClassHandle(
       RTI::RTIambassador* amb) const;

 //get the number of parameters
 virtual int numParameters() const;

 //name of the interaction
 virtual const char* name() const;

 //print the interaction's data
 virtual void printData() const;

 //accessors
 const GtTeamType senderTeam() const;
 const DtGameEntityIdentifier& entityID() const;
 const GtBodyType entityBodyType() const;
   const GtTeamType entityTeam() const;
 const char* label() const;

 //mutators
 void SetSenderTeam( const GtTeamType senderTeam );
```

```
void SetBodyType(const GtBodyType body);
void SetEntityTeam(const GtTeamType team);
void SetLabel(LPCSTR szLabel);

public:
 //get the interacton class handle
 static RTI::InteractionClassHandle classHandle(
       RTI::RTIambassador* amb);

 //create a SphCtrlIVISEntityLabelInteraction.  Caller responsbile for deletion
 static DtInteraction* create(RTI::InteractionClassHandle h,
       const RTI::ParameterHandleValuePairSet& p,
       RTI::RTIambassador* amb);

 // Add (register)/Remove (deregister) function to be called
 // when the interaction occurs.
 static void addCallback(DtExerciseConn* conn,
       SphCtrlIVISEntityLabelInteractionCb cb, void* usr);

 static void removeCallback(DtExerciseConn* conn,
       SphCtrlIVISEntityLabelInteractionCb cb, void* usr);

protected:
 //our actual data to be sent/received
 GtTeamType              m_SenderTeam;
 DtGameEntityIdentifier  m_entityID;
 GtBodyType      m_BodyType;

  GtTeamType      m_entityTeam;

  char            m_szLabel[CTRL_IVIS_ENTITY_LABEL_LEN + 1];

protected:
 //used as return value by phvps - does not store state
 static RTI::ParameterHandleValuePairSet* thePvList;
};
```

3.1.9 SphCtrlIVISEntityMoveToInteraction

```
typedef void (*SphCtrlIVISEntityMoveToInteractionCb)(SphCtrlIVISEntityMoveToInteraction* inter, void*
usr);

class SphCtrlIVISEntityMoveToInteraction : public DtInteraction
{
public:
 SphCtrlIVISEntityMoveToInteraction();

 SphCtrlIVISEntityMoveToInteraction(const RTI::ParameterHandleValuePairSet& pvList,
       RTI::RTIambassador* amb);

 SphCtrlIVISEntityMoveToInteraction(GtTeamType senderTeam,
       DtGameEntityIdentifier& entityID,
       double x,
       double y);

 virtual ~SphCtrlIVISEntityMoveToInteraction();
```

```
//get the paramater handle value pair set for the interaction
const RTI::ParameterHandleValuePairSet& phvps(
      RTI::RTIambassador* amb) const;

//set the values from parameter value list
virtual void setFromPhvps(const RTI::ParameterHandleValuePairSet& pvList,
      RTI::RTIambassador* amb);

//get the interaction class handle
virtual RTI::InteractionClassHandle interactionClassHandle(
      RTI::RTIambassador* amb) const;

//get the number of parameters
virtual int numParameters() const;

//name of the interaction
virtual const char* name() const;

//print the interaction's data
virtual void printData() const;

//accessors
const GtTeamType senderTeam() const;
const DtGameEntityIdentifier& entityID() const;
  const double X() const;
  const double Y() const;

//mutators
void SetSenderTeam( const GtTeamType senderTeam );
void SetEntityID(const DtGameEntityIdentifier& id);

  void SetX( const double x );

  void SetY( const double y );

public:
 //get the interacton class handle
 static RTI::InteractionClassHandle classHandle(
      RTI::RTIambassador* amb);

 //create a SphCtrlIVISEntityMoveToInteraction.  Caller responsbile for deletion
 static DtInteraction* create(RTI::InteractionClassHandle h,
      const RTI::ParameterHandleValuePairSet& p,
      RTI::RTIambassador* amb);

 // Add (register)/Remove (deregister) function to be called
 // when the interaction occurs.
 static void addCallback(DtExerciseConn* conn,
      SphCtrlIVISEntityMoveToInteractionCb cb, void* usr);

 static void removeCallback(DtExerciseConn* conn,
      SphCtrlIVISEntityMoveToInteractionCb cb, void* usr);

protected:
 //our actual data to be sent/received
 GtTeamType            m_SenderTeam;
 DtGameEntityIdentifier m_entityID;
 double                m_x;
 double                m_y;

protected:
```

```
 static RTI::ParameterHandleValuePairSet* thePvList;
};
```

3.1.10 SphCtrlIVISEntityOrientationInteraction

```
typedef void (*SphCtrlIVISEntityOrientationInteractionCb)(SphCtrlIVISEntityOrientationInteraction* inter,
void* usr);

class SphCtrlIVISEntityOrientationInteraction : public DtInteraction
{
public:
 SphCtrlIVISEntityOrientationInteraction();

 SphCtrlIVISEntityOrientationInteraction(const RTI::ParameterHandleValuePairSet& pvList,
       RTI::RTIambassador* amb);

 SphCtrlIVISEntityOrientationInteraction(GtTeamType senderTeam,
       DtGameEntityIdentifier& entityID,
       int heading);

 virtual ~SphCtrlIVISEntityOrientationInteraction();

 //get the paramater handle value pair set for the interaction
 const RTI::ParameterHandleValuePairSet& phvps(
       RTI::RTIambassador* amb) const;

 //set the values from parameter value list
 virtual void setFromPhvps(const RTI::ParameterHandleValuePairSet& pvList,

       RTI::RTIambassador* amb);


 //get the interaction class handle
 virtual RTI::InteractionClassHandle interactionClassHandle(
       RTI::RTIambassador* amb) const;

 //get the number of parameters
 virtual int numParameters() const;

 //name of the interaction
 virtual const char* name() const;

 //print the interaction's data
 virtual void printData() const;

 //accessors
 const GtTeamType senderTeam() const;
 const DtGameEntityIdentifier& entityID() const;
   const int heading() const;

 //mutators
 void SetSenderTeam( const GtTeamType senderTeam );
 void SetEntityID(const DtGameEntityIdentifier& id);
   void SetHeading( const int heading );

public:
```

```
static RTI::InteractionClassHandle classHandle(
      RTI::RTIambassador* amb);

//create a SphCtrlIVISEntityOrientationInteraction.  Caller responsbile for deletion
static DtInteraction* create(RTI::InteractionClassHandle h,
      const RTI::ParameterHandleValuePairSet& p,
      RTI::RTIambassador* amb);

// Add (register)/Remove (deregister) function to be called
// when the interaction occurs.
static void addCallback(DtExerciseConn* conn,
      SphCtrlIVISEntityOrientationInteractionCb cb, void* usr);

static void removeCallback(DtExerciseConn* conn,
      SphCtrlIVISEntityOrientationInteractionCb cb, void* usr);

protected:
 //our actual data to be sent/received
 GtTeamType              m_SenderTeam;
 DtGameEntityIdentifier  m_entityID;
 int                     m_Heading;

protected:
 //used as return value by phvps - does not store state
 static RTI::ParameterHandleValuePairSet* thePvList;
};
```

3.1.11 SphCtrlIVISEntityROEInteraction

```
typedef void (*SphCtrlIVISEntityROEInteractionCb)(SphCtrlIVISEntityROEInteraction* inter, void* usr);


class SphCtrlIVISEntityROEInteraction : public DtInteraction

{
public:
 SphCtrlIVISEntityROEInteraction();

 SphCtrlIVISEntityROEInteraction(const RTI::ParameterHandleValuePairSet& pvList,
      RTI::RTIambassador* amb);

 SphCtrlIVISEntityROEInteraction(GtTeamType senderTeam,
      DtGameEntityIdentifier& entityID,
      GtRuleOfEngagementArg   ROE);

 virtual ~SphCtrlIVISEntityROEInteraction();

 //get the paramater handle value pair set for the interaction
 const RTI::ParameterHandleValuePairSet& phvps(
      RTI::RTIambassador* amb) const;

 //set the values from parameter value list
 virtual void setFromPhvps(const RTI::ParameterHandleValuePairSet& pvList,
      RTI::RTIambassador* amb);

 //get the interaction class handle
```

```
        RTI::RTIambassador* amb) const;

    //get the number of parameters
    virtual int numParameters() const;

    //name of the interaction
    virtual const char* name() const;

    //print the interaction's data
    virtual void printData() const;

    //accessors
    const GtTeamType senderTeam() const;
    const DtGameEntityIdentifier& entityID() const;
      const GtRuleOfEngagementArg ROE() const;

    //mutators
    void SetSenderTeam( const GtTeamType senderTeam );
    void SetEntityID(const DtGameEntityIdentifier& id);
      void SetROE( const GtRuleOfEngagementArg roe );

  public:
    //get the interacton class handle
    static RTI::InteractionClassHandle classHandle(
          RTI::RTIambassador* amb);

    //create a SphCtrlIVISEntityROEInteraction.  Caller responsbile for deletion
    static DtInteraction* create(RTI::InteractionClassHandle h,
          const RTI::ParameterHandleValuePairSet& p,
          RTI::RTIambassador* amb);

    // Add (register)/Remove (deregister) function to be called
    // when the interaction occurs.
    static void addCallback(DtExerciseConn* conn,
          SphCtrlIVISEntityROEInteractionCb cb, void* usr);

    static void removeCallback(DtExerciseConn* conn,
          SphCtrlIVISEntityROEInteractionCb cb, void* usr);



  protected:
    //our actual data to be sent/received
    GtTeamType                  m_SenderTeam;
    DtGameEntityIdentifier      m_entityID;
    GtRuleOfEngagementArg   m_ROE;

  protected:
    //used as return value by phvps - does not store state
    static RTI::ParameterHandleValuePairSet* thePvList;
  };
```

3.1.12 DtMachineGunFireInteraction

```
typedef void (*DtMachineGunFireInteractionCb)(DtMachineGunFireInteraction* inter, void* usr);
```

```
{
public:

  // default constructor
  DtMachineGunFireInteraction();

  // constructor
  DtMachineGunFireInteraction( const RTI::ParameterHandleValuePairSet& pvList,
                               RTI::RTIambassador* amb );

  // destructor
  virtual ~DtMachineGunFireInteraction();

  // copy constructor
  DtMachineGunFireInteraction(const DtMachineGunFireInteraction& orig);

  // assignment operator
  DtMachineGunFireInteraction& operator=(const DtMachineGunFireInteraction& orig);

  // Get the parameter handle value pair set for the interaction.
  const RTI::ParameterHandleValuePairSet& phvps( RTI::RTIambassador* amb ) const;

  // Decode into this object from parameter value list.
  virtual void setFromPhvps( const RTI::ParameterHandleValuePairSet& pvList,
                             RTI::RTIambassador* amb );

  // Get the interaction class handle.
  virtual RTI::InteractionClassHandle interactionClassHandle( RTI::RTIambassador* amb ) const;

  // Get the number of parameters.
  virtual int numParameters() const;

  // Print the interaction's data.
  virtual void printData() const;

  // Set/Get the attacker identifier.
  void setAttackerId( const DtGameEntityIdentifier& id );
  virtual const DtGameEntityIdentifier& attackerId() const;

  void setHLAAttackerId( const DtObjectId& id );
  virtual const DtObjectId& HLAattackerId() const;



  // Set/Get the starting location of the detonation in entity coordinates.
  void setStartLocation( DtConstVector location );
  void setStartLocation( DtNetWorldCoordinates loc );
  DtConstVector startLocation() const;

  // Set/Get the ending location of the detonation in entity coordinates.
  void setEndLocation( DtConstVector location );
  void setEndLocation( DtNetWorldCoordinates loc );
  DtConstVector endLocation() const;

  // Set/Get the burst descriptor
  void setBurst( const DtGameBurstDescriptor& burst );
  const DtGameBurstDescriptor& burst() const;

  // Name of the interaction
  virtual const char* name() const;
```

```
  // Get the interaction class handle.
  static RTI::InteractionClassHandle classHandle( RTI::RTIambassador* amb );

  // Create a DtMachineGunFireInteraction.
  // Caller is reponsible for deletion.
  static DtInteraction* create( RTI::InteractionClassHandle h,
                                const RTI::ParameterHandleValuePairSet& p,
                                RTI::RTIambassador* amb );

  // Add (register)/Remove (deregister) function to be called when
  // interaction occurs.
  static void addCallback( DtExerciseConn* conn, DtMachineGunFireInteractionCb cb,
                           void* usr);

  static void removeCallback( DtExerciseConn* conn, DtMachineGunFireInteractionCb cb,
                              void* usr);

protected:
  // Decode into this object from parameter value list.
  // Non-virtual since meant to be called from ctor.
  void internalSetFromPhvps( const RTI::ParameterHandleValuePairSet& pvList,
                             RTI::RTIambassador* amb );

protected:
   DtGameEntityIdentifier myAttackerId;
 DtObjectId myHLAAttackerId;
   DtVector myStart;
   DtVector myEnd;
   DtGameBurstDescriptor myBurst;

protected:
  static RTI::ParameterHandleValuePairSet* thePvList;
};
```

3.1.13 DtMachineGunDetonationInteraction

```
typedef void (*DtMachineGunDetonationInteractionCb)(DtMachineGunDetonationInteraction* inter, void*
usr);



class DtMachineGunDetonationInteraction : public DtInteraction
{
public:

  // default constructor
  DtMachineGunDetonationInteraction();

  // constructor
  DtMachineGunDetonationInteraction( const RTI::ParameterHandleValuePairSet& pvList,
                                     RTI::RTIambassador* amb );

  // copy constructor
  DtMachineGunDetonationInteraction(const DtMachineGunDetonationInteraction& orig);
```

```
virtual ~DtMachineGunDetonationInteraction();

// assignment operator
DtMachineGunDetonationInteraction& operator=(const DtMachineGunDetonationInteraction& other);

// Get the parameter handle value pair set for the interaction.
const RTI::ParameterHandleValuePairSet& phvps( RTI::RTIambassador* amb ) const;

// Decode into this object from parameter value list.
virtual void setFromPhvps( const RTI::ParameterHandleValuePairSet& pvList,
                           RTI::RTIambassador* amb );

// Get the interaction class handle.
virtual RTI::InteractionClassHandle interactionClassHandle( RTI::RTIambassador* amb ) const;

// Get the number of parameters.
virtual int numParameters() const;

// Print the interaction's data.
virtual void printData() const;

// Set/Get the attacker identifier.
void setAttackerId( const DtGameEntityIdentifier& id );
virtual const DtGameEntityIdentifier& attackerId() const;

void setHLAAttackerId( const DtObjectId& id );
virtual const DtObjectId& HLAattackerId() const;

// Set/Get the target identifier.
void setTargetId( const DtGameEntityIdentifier& id );
virtual const DtGameEntityIdentifier& targetId() const;

void setHLATargetId( const DtObjectId& id );
virtual const DtObjectId& HLAtargetId() const;

// Set/Get the starting location of the detonation in entity coordinates.
void setStartLocation( DtConstVector location );
void setStartLocation(DtNetWorldCoordinates loc);
DtConstVector startLocation() const;

// Set/Get the ending location of the detonation in entity coordinates.
void setEndLocation( DtConstVector location );
void setEndLocation(DtNetWorldCoordinates loc);
DtConstVector endLocation() const;

// Set/Get the burst descriptor

void setBurst( const DtGameBurstDescriptor& burst );

const DtGameBurstDescriptor& burst() const;

// Set/Get the entity side
void setEntitySide( DtEntitySide side );
DtEntitySide entitySide() const;

// Set/Get the detonation result
void setResult( DtDetonationResult result );
DtDetonationResult result() const;

// Name of the interaction
virtual const char* name() const;
```

```
public:

  // Get the interaction class handle.

  static RTI::InteractionClassHandle classHandle( RTI::RTIambassador* amb );

  // Create a DtMachineGunDetonationInteraction.
  // Caller is reponsible for deletion.

  static DtInteraction* create( RTI::InteractionClassHandle h,
                                const RTI::ParameterHandleValuePairSet& p,
                                RTI::RTIambassador* amb );

  // Add (register)/Remove (deregister) function to be called when
  // interaction occurs.

  static void addCallback( DtExerciseConn* conn, DtMachineGunDetonationInteractionCb cb,
                           void* usr);

  static void removeCallback( DtExerciseConn* conn, DtMachineGunDetonationInteractionCb cb,
                              void* usr);

protected:

  // Decode into this object from parameter value list.
  // Non-virtual since meant to be called from ctor.

  void internalSetFromPhvps( const RTI::ParameterHandleValuePairSet& pvList,
                             RTI::RTIambassador* amb );

protected:

   DtGameEntityIdentifier myAttackerId;
   DtGameEntityIdentifier myTargetId;

 DtObjectId myHLAAttackerId;
 DtObjectId myHLATargetId;

   DtVector myStart;
   DtVector myEnd;
   DtGameBurstDescriptor myBurst;
   DtEntitySide mySide;
   DtDetonationResult myResult;

protected:
 static RTI::ParameterHandleValuePairSet* thePvList;
};
```

3.1.14 GntByteMoverInteraction

This interaction is used to move transfer mission files and mission briefings from one federate to another.

```
typedef void (*GntByteMoverInteractionCb)(GntByteMoverInteraction* inter, void *usr);

class GntByteMoverInteraction : public DtInteraction
{
```

```
GntByteMoverInteraction();

GntByteMoverInteraction(const RTI::ParameterHandleValuePairSet& pvList,
      RTI::RTIambassador* amb);

virtual ~GntByteMoverInteraction();

//get the paramater handle value pair set for the interaction
const RTI::ParameterHandleValuePairSet& phvps(
      RTI::RTIambassador* amb) const;

//set the values from parameter value list
virtual void setFromPhvps(const RTI::ParameterHandleValuePairSet& pvList,
      RTI::RTIambassador* amb);

//get the interaction class handle
virtual RTI::InteractionClassHandle interactionClassHandle(
      RTI::RTIambassador* amb) const;

//get the number of parameters
virtual int numParameters() const;

//name of the interaction
virtual const char* name() const;

//print the interaction's data
virtual void printData() const;

//get/set the NumAlloc parameter
int getNumAlloc() const;
void setNumAlloc(int num);

//get/set the Buffer parameter
const void* getBuffer() const;
void setBuffer(const void* buff, const int NumBytes);

public:
 //get the interacton class handle
 static RTI::InteractionClassHandle classHandle(
      RTI::RTIambassador* amb);

 //create a GntByteMoverInteraction.  Caller responsbile for deletion
 static DtInteraction* create(RTI::InteractionClassHandle h,
      const RTI::ParameterHandleValuePairSet& p,
      RTI::RTIambassador* amb);

 // Add (register)/Remove (deregister) function to be called
 // when the interaction occurs.

 static void addCallback(DtExerciseConn* conn,

      GntByteMoverInteractionCb cb, void* usr);

 static void removeCallback(DtExerciseConn* conn,
      GntByteMoverInteractionCb cb, void* usr);

protected:
 //our actual data to be sent/received
 int         m_iNumAlloc;
 void* m_pBuffer;
```

```
 //used as return value by phvps - does not store state
 static RTI::ParameterHandleValuePairSet* thePvList;

 void clearBuffer();
};
```

3.1.15 GntControlMessageInteraction

```
typedef void (*GntControlMessageInteractionCb)(GntControlMessageInteraction* inter, void *usr);

class GntControlMessageInteraction : public GntByteMoverInteraction
{
public:
 GntControlMessageInteraction();

 GntControlMessageInteraction(const RTI::ParameterHandleValuePairSet& pvList,
      RTI::RTIambassador* amb);

 virtual ~GntControlMessageInteraction();

 //set the values from parameter value list
 virtual void setFromPhvps(const RTI::ParameterHandleValuePairSet& pvList,
      RTI::RTIambassador* amb);

 //get the interaction class handle
 virtual RTI::InteractionClassHandle interactionClassHandle(
      RTI::RTIambassador* amb) const;

 //get the number of parameters
 virtual int numParameters() const;

 //name of the interaction
 virtual const char* name() const;

 //print the interaction's data
 virtual void printData() const;

 void SetBufferAsString(const char* str);

 //accessors
 const unsigned long GetCtrlID() const;

 //mutators
 void SetCtrlID(unsigned long id);

public:

 //get the interacton class handle

 static RTI::InteractionClassHandle classHandle(
      RTI::RTIambassador* amb);

 //create a GntControlMessageInteraction.  Caller responsbile for deletion
 static DtInteraction* create(RTI::InteractionClassHandle h,
      const RTI::ParameterHandleValuePairSet& p,
      RTI::RTIambassador* amb);
```

```
 // Add (register)/Remove (deregister) function to be called
 // when the interaction occurs.
 static void addCallback(DtExerciseConn* conn,
       GntControlMessageInteractionCb cb, void* usr);

 static void removeCallback(DtExerciseConn* conn,
       GntControlMessageInteractionCb cb, void* usr);

protected:
 unsigned long   m_CtrlId;

protected:
 //used as return value by phvps - does not store state
 static RTI::ParameterHandleValuePairSet* thePvList;
};
```

3.2 Game Entity Identifier Conversion Functions

```
#define             PLAYERDIV     100000

const DtGameEntityIdentifier& Cv2GEId(const DtObjectId& id)
{
 static DtGameEntityIdentifier GEId;

 DtU16 nPlayerId = id / PLAYERDIV;
 DtU16 nEntityNum = id - (nPlayerId * PLAYERDIV);

 GEId.init(nPlayerId, nEntityNum);
 return GEId;
}


const DtObjectId& Cv2OID( const DtGameEntityIdentifier& GEId )
{
 static DtObjectId OID;

 RTI::ObjectID Id = (GEId.playerId() * PLAYERDIV) + GEId.entityNum();
 OID = Id;
 return OID;
}
```

Section 4 Appendix B Federation File modifications

The following excerpt contains the modifications made to the Vr-Link.fed file, which incorporates the functionality of the RPR FOM.

```
(class ByteMover FED_RELIABLE FED_RECEIVE
 (parameter NumAlloc)
 (parameter Buffer)
  )
  (class GntControlMessageInteraction FED_RELIABLE FED_RECEIVE
 (parameter CtrlId)
 (parameter NumAlloc)
 (parameter Buffer)
  )
  (class SphCommentInteraction FED_RELIABLE FED_RECEIVE
 (parameter TeamFilter)
 (parameter ChatMessage)
 (parameter SenderId)
 (parameter ReceiverId)
  )
  (class SphChatInteraction FED_RELIABLE FED_RECEIVE
 (parameter TeamFilter)
 (parameter ChatMessage)
 (parameter SenderId)
 (parameter ReceiverId)
  )
  (class SphCtrlIVISAirAttackInteraction FED_RELIABLE FED_RECEIVE
 (parameter SenderTeam)
 (parameter X)
 (parameter Y)
  )
  (class SphCtrlIVISAirReconInteraction FED_RELIABLE FED_RECEIVE
 (parameter SenderTeam)
 (parameter X)
 (parameter Y)
  )
  (class SphCtrlIVISArtilleryInteraction FED_RELIABLE FED_RECEIVE
 (parameter SenderTeam)
 (parameter X)
 (parameter Y)
  )
  (class SphCtrlIVISEntityEngageInteraction FED_RELIABLE FED_RECEIVE
 (parameter SenderTeam)
 (parameter EntityId)
 (parameter TargetId)
  )
  (class SphCtrlIVISEntityKillLabelInteraction FED_RELIABLE FED_RECEIVE
 (parameter SenderTeam)
 (parameter EntityId)
  )
  (class SphCtrlIVISEntityLabelInteraction FED_RELIABLE FED_RECEIVE
 (parameter SenderTeam)
 (parameter EntityId)
 (parameter BodyType)
 (parameter EntityTeam)
```

```
(parameter Label)
 )

 (class SphCtrlIVISEntityMoveToInteraction FED_RELIABLE FED_RECEIVE
(parameter SenderTeam)
    (parameter EntityId)
(parameter X)
(parameter Y)
 )
 (class SphCtrlIVISEntityOrientationInteraction FED_RELIABLE FED_RECEIVE
(parameter SenderTeam)
    (parameter EntityId)
(parameter Orientation)
 )
 (class SphCtrlIVISEntityROEInteraction FED_RELIABLE FED_RECEIVE
(parameter SenderTeam)
    (parameter EntityId)
(parameter ROE)
 )
 (class MachineGunDetonationInteraction FED_RELIABLE FED_RECEIVE
(parameter AttackerID)
    (parameter TargetID)
(parameter StartLocation)
(parameter EndLocation)
(parameter Burst)
(parameter EntitySide)
(parameter Result)
 )
 (class MachineGunFireInteraction FED_RELIABLE FED_RECEIVE
(parameter AttackerID)
    (parameter StartLocation)
(parameter EndLocation)
(parameter Burst)
 )
```

Section 5 Installing the HLA Spearhead Demo CD

The Phase I effort of this SBIR has an install program that will install both HLA Spearhead and the RTI. The process requires a few simple steps that the user must perform.

5.1 System Requirements

Windows 95 (will not run under Windows NT)

Pentium 120 MHz or higher based system, Pentium 166Mhz or higher recommended.

16MB RAM minimum, 32MB recommended

Windows 95 compatible video card capable of 16 BPP resolution (64K colors)

Windows 95 compatible 16 or 32 bit sound card (e.g., Sound Blaster)

Windows 95 compatible LAN card with TCP/IP support installed.

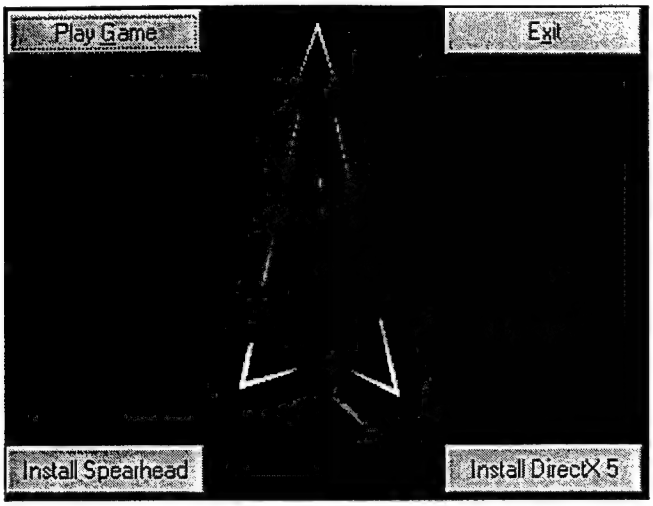
280MB Free hard disk space

Microsoft Direct X version 5.0 (supplied on HLA Spearhead CD-ROM)

5.2 Installation Steps

1. The first step is to install Microsoft Direct X, since Spearhead cannot run without it. Direct X technology provides specialized interfaces to PC hardware so games can run an optimal speed under Windows 95. Perform these steps to install Direct X.

Insert the HLA Spearhead CD-ROM in your CD-ROM drive. The CD-ROM uses the auto start feature of Windows 95, which means you automatically see the following screen after a few seconds.



If Direct X 5 is not installed on your system, then you should click the Install **Direct X 5** button. When Direct X is done installing, you must **re-boot** your system.

2. The second step is to install Spearhead. You can click over the **Install Spearhead** button, or run the **SETUP.EXE** program on the CD-ROM. Just follow the all the default prompts. Once the setup is complete, you should have an **HLA Spearhead** folder in your Start menu. Setup will create the following menu items in the HLA Spearhead folder:

   - Control Setup
   - Spearhead
   - RTI Exec
   - Mission Editor
   - ReadMe
   - Uninstall Spearhead

3. If you have a **3DFX** based graphics accelerator, such as the Voodoo 3D or Monster 3D boards, make sure you have the latest version of the **Glide Drivers** installed. If you are not sure about this, there is a **3DFX_Drivers** folder on the CD-ROM that contains 3DFX Glide drivers that will work with Spearhead. You can install the Glide drivers from that directory, the files are as follows:

   glide231.exe – Voodoo Graphics Glide 2.31 drivers

   monster_3dfx108.exe – Monster board drivers

   ordchid_3dfx.exe – Orchid board

Please install the drivers appropriate to your card. If you do not have a 3DFX board, please ignore the above.

4. You must now modify your **AUTOEXEC.BAT** file since the RTI requires that some environment variables are set. Add the following lines to your AUTOEXEC.BAT file. Please note that you will need to change the C:\ if you installed to another drive such as D:\.

```
SET RTI_HOME=C:\SPEARHEAD\RTI
SET RTI_CONFIG=%RTI_HOME%\config
```

5. Before running HLA Spearhead, you must configure the RTI to work on your machine. HLA Spearhead WILL NOT RUN until you do so. Edit the following file: **C:\Spearhead\RTI\Config\RTI.RID**

Search for the following text:

RTI_EXEC_HOST YOUR_MACHINE_NAME

Change the YOUR_MACHINE_NAME text to the actual name of your machine. To obtain your machine name, go to the properties tab of **Network Neighborhood** and click on the **Identification** Tab. Your actual machine name will appear in the **Computer Name** field.

6. Also, if you are attempting to play multiplayer games in Spearhead on systems different **LAN subnets**, you will need to add the following line to the AUTOEXEC.BAT file. Please change the netmask to match your LAN subnet.

```
SET DTNETMASK=255.255.255.192
```

For the sake of simplicity, it is recommended that you attempt Spearhead multiplayer games on systems within the same LAN subnet.

Section 6 Section 6 Running the HLA Spearhead Demo

Before running the HLA Spearhead demo, it is strongly suggested that you read the following attached documents:

- **Quick Reference Guide** – This will give you a quick outline of all the keys and controls in Spearhead. From this document you will have an overview of what features are present in the game and how to access them.

- **Spearhead Tank View and Controls** – This document will show you all the views in the game, such as driver, gunner, commander, IVIS, and how to use them. Familiarize yourself with this document since it contains important information about each view, its features and related keyboard/joystick controls.

- **Keyboard Layout** – keep this sheet handy for quick reference when playing the game.

- **Spearhead Manual** - This is the full documentation for Spearhead, it also covers the Mission Editor.

6.1 Running the RTI Exec

You must run the RTI Exec before attempting either a single player or multiplayer game in HLA Spearhead. **HLA Spearhead will not run unless the RTI Exec is running first!**

To run the RTI Exec, go to the Windows **Start** menu, then go to the **HLA Spearhead\RTI Exec** file folder and run the RTI Exec program. You should see a DOS console window with the RTI Exec running inside it. If the RTI is running properly, you should see a message of the form:

Log Acceptor open on 0x00004070

Note: The number will change on your system.

6.2 Running HLA Spearhead

Once the RTI Exec is running, you can now run HLA Spearhead. To run it, go to the Windows **Start** menu, then go to the **HLA Spearhead** file folder and run the Spearhead program.

You should see some introductory videos, you can short circuit these videos and proceed directly to the main menu by pressing the Space Bar.

6.2.1 Run A Single Player Game First To Test HLA Spearhead

Once you are at the main menu, attempt to play a single player game first. This is an easy test since only one PC is required. When you are at the main menu, use the mouse to click on the following buttons:

- Click the **Single Player** button
- Click over the **Lesson1_MainGun** mission so it is hi-lighted.
- Click the **Start Game Easy** button

The mission will now load and you should see a **FEDEX** DOS Console window appear over the game. Once the game is loaded, **click the mouse on the Spearhead screen so it has the input focus from the user**.

- Press the **F6** button so you are in Gunner mode
- Use the **Arrow Keys** to slew the turret and press the **Space Bar** to fire the main gun.

Please refer to the accompanying documents for other features such as driving the tank, etc.

- Once you are done, press **Ctrl-X** to exit the game.
- You will see a video, just press the **Space Bar** to end the video.
- Click the **EXIT** button to return to the main menu.

You can now play other single player missions or move on to playing a multiplayer game.

6.2.2 Playing a Simple Multiplayer Game

In order to play a multiplayer game, you will need to install HLA Spearhead on two or more machines. **Only one machine needs to run the RTI Exec. Please refer to the accompanying Spearhead manual for additional instructions on playing multiplayer games.**

You do not need the Spearhead CD-ROM in order to play, since all the files are copied on to the hard disk.

Select a machine to "host" the mission and another to join. We will try a simple two player game. You should get the host running before attempting to join a mission.

NOTE: DUE TO CURRENT RTI RESTRICTIONS, YOU CAN ONLY PLAN LAN GAMES. INTERNET AND HEAD TO HEAD MODEM GAMES WILL NOT WORK AND WILL CAUSE THE SYSTEM TO CRASH.

6.2.2.1 Instructions for the Host Machine

Run The RTI Exec program

Run HLA Spearhead

When you are at the main menu, use the mouse to click on the following buttons:

Click the **Multi Player** button

Click the **IPX/LAN** Button

Enter your name (i.e., Mark)

Click the **Host** button

Click over the **Multiplayer00_1on1Test** mission

Click the **Create Def. Host** button

Enter a mission name (i.e. Test)

Wait for the joiner

Click the **Start Game** button

6.2.2.2 Instructions for the Joiner Machine

Run HLA Spearhead

When you are at the main menu, use the mouse to click on the following buttons:

Click the **Multi Player** button

Click the **IPX/LAN** Button

Enter your name (i.e., Mark)

Click the **Join Defensive** button

Click the **OK** button

Wait for the host to start the game.

Once the game is done loading for both machines, you should be facing one another. You can try shooting, driving, etc. Press **Ctrl-X** when you want to exit. You will be returned to either the host or joiner screen. Try experimenting with other multiplayer missions, such as **Multiplayer00_2on2Test, Multiplayer00_3on3Test, etc.**

6.3 Known Problems Running HLA Spearhead Phase 1

Game resets may not work and could cause the RTI to crash. It is recommended that you completely exit Spearhead (press Alt-F4 or the Exit button from the main menu) before attempting to run other single or multiplayer missions. This applies to both the host and joiner machines. The cause of this problem was lack of time to code a robust reset sequence in HLA.

Articulated parts do not update on remote entities. While playing multiplayer games you will notice that articulated body parts, such as turrets, will not update their positions for remote entities. Only local entities will have updating functioning for art parts. This problem does not affect single player games.

On occasion, the RTI may refuse to connect with Spearhead and throw an exception that will cause Spearhead to terminate. When this happens, shut down the RTI, re-start the RTI and Spearhead. This is relatively rare occurrence.

Missions containing minefields will cause the RTI to crash since Spearhead assigns and ID of 0 to minefields. The RTI cannot handle and ID of 0. This problem has been fixed in later releases of Spearhead.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave Blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	1998 05 15	Final 1998-10-1 to 1998-04-31

4. TITLE AND SUBTITLE
Integrating HLA into the Spearhead Game
OSD97-003

5. FUNDING NUMBERS
VI67004-97-C-0046

6. AUTHOR(S)
Robert Faucher

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
MäK Technologies
185 Alewife Brook Parkway
Cambridge, MA 01238

8. PERFORMING ORGANIZATION REPORT NUMBER
TR-OSD97003-981515.1

9. SPONSORING/MONITORING AGENGY NAME(S) AND ADDRESS(ES)
U.S. Army Simulation, Training and Instrumentation Command
Mark E. McAuliffe

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLIMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Not restricted

12b. DISTRIBUTION CODE

13. ABSTRACT
Final report

14. SUBJECT TERMS

15. NUMBER OF PAGES
44

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	